ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

The Design, Implementation and Performance Evaluation for Middle Tier Based Replication Scheme

Chintareddy Venkat Ramana Reddy
M.Tech,
Hyderabad.

Madireddy Vijay Reddy
M.Tech,
Hyderabad.

Abstract:

Recent client-server distributed computing systems may be seen as implementations of N-tier architectures. Typically, the first tier consists of client applications containing browsers, with the remaining tiers deployed within an enterprise representing the server side; the second tier (Web tier) consists of web servers that receive requests from clients and pass on the requests to specific applications residing in the third tier (middle tier) consisting of application servers where the computations implementing the business logic are performed; and the fourth tier (database/back-end tier) contains databases that maintain persistent data for applications. Existing application server that are capable to handle entire architecture Web logic , JBOSS ,Web sphere .

A specification model should be develop that aloes approach to build N-tire architecture application. The performance of the existing application server is proportional to N-tier application developer's expose and skills in technology this are the limitations of the existing application. Is Enhancement of application server that would generate an N-tier architecture template based on the given requirements so, develops enhance their template to fulfill there business requirements. The focus of this project work has been to combine existing techniques to provide a complete working solution on a real platform. Our techniques for replicating the database tier and that for replicating the middle tier are not dependent on each other and can be used together or separately without any difficulties.

Index Terms:

Application servers, availability, Enterprise Java Beans, fault tolerance, middleware, replication, transactions.

INTRODUCTION:

The goal is to facilitate and standardize the development, deployment, and assembling of application components. Application components are deployable on the J2EE application server (AS) middleware. The middleware contains a set of services for naming, messaging, transactions, persistence, security, logging, and so on. However, the specification does not impose any implementation model in the construction of the middleware. This is left to the AS providers. In this paper, we show that adopting a component-based architecture of the middleware improves the management functions. Indeed, a management based on the knowledge of the system architecture allows to handle the interdependence between the system components and enables performing reconfiguration tasks. The server embeds different Software from different Object-Web members. Indeed, the services that offer the non-functional properties to the J2EE applications encapsulate software from different developers.

This involves challenging management needs to handle this heterogeneity, particularly handling the versioning issues. The distribution aspect of the AS are not addressed by J2EE standard. In addition, in a distributed environment, as a cluster, new levels of management abstractions are needed. For example, it is interesting to expose the Web tier, which contains a set of Web containers, as a single virtual Web container to the manager to aggregate and simplify some administration tasks. For example, instead of deploying a same application on each Web container replica, the manager needs a simpler operation: deploying the application on the tier. This hides the distribution and the modifications in the tier that will be handled at a lower level. Furthermore, a J2EE cluster can run different J2EE applications. It is important to group the resources used by an application under a management unit to control the application performance separately.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

This architecture also allows flexible configuration using clustering for improved performance and scalability. Availability measures, such as replication, can be introduced in each tier in an application specific manner. In a typical n-tier system, such as illustrated in the interactions between clients and the web tier are performed across the Internet. The infrastructures supporting these interactions are generally beyond the direct control of an application service provider. The middle and the database tiers are the most important, as it is on these tiers that the computations are performed and persistency provided. These two tiers are considered in this paper.

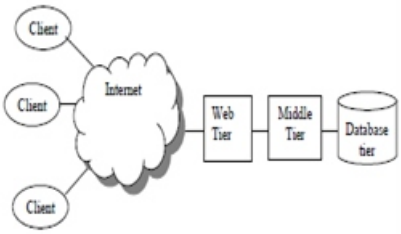

Fig.: N-tier architecture

Further , availability measures, such as replication, can be introduced in each tier in an application specific manner. However, incorporation of availability measures in a multi-tier system poses challenging system design problems of integrating open, non proprietary solutions to transparent failover, exactly once execution of client requests, non-blocking transaction processing and an ability to work with clusters. This paper describes how replication for availability can be incorporated within the middle and back-end tiers meeting all these challenges. The paper develops an approach that requires enhancements to the middle tier only for supporting replication of both the middleware backend tiers. The design, implementation and performance evaluation of such a middle tier based replication scheme for multi-database transactions on a widely deployed open source application server (JBoss) are presented.The problem of exposing the relations between the AS services is also raised . A management system is built on top of the AS (JBoss) to express the dependencies between services. Our approach is to modify the middleware itself and adopt a modular architecture where dependencies are explicit. SmartFrog is a deployment framework that is applicable to the ignition and deployment of AS in a cluster environment. The system proposes a component model that allows encapsulating the cluster parts (Apache, Tomcat, AS etc.) and expressing the dependencies between these parts.

RELATED WORK:

Group communications plays a pivotal role in the evolution of availability solutions we see in a number of replication management schemes. The classic text discusses replicated data management techniques that go hand in hand with transactions with the Arjuna system demonstrating the use of transactions in the support of replicated transactional objects . With the advent of object-oriented standards for distributed computing, a number of replication based solutions were developed, specifically in the area of CORBA making extensive use of group communications to ensure consistency of replicas. Recent works have seen transactions and group communications applied to Web Services to aid recoverability. For example, configurable durability to allow data persistence to survive crash failure and utilizing group communications to provide replication schemes for increased availability . Although there are similar techniques used in both approaches (as Web Services are often implemented using n-tier environments), the focus in this paper is an engineered solution for availability in transactional n-tier systems.In the rest of this section we will examine prior work on availability measures for transactional data (objects) in n-tier architectures, beginning with current industrial practice for EJB application servers. We also describe how our work differs from the relevant research work reported so far.

The key requirement here is to ensure exactly once execution of transactional requests. The interplay between replication and exactly once execution within the context of multi-tier architectures is examined in , whilst describes how replication and transactions can be incorporated into three-tier CORBA architectures. The approachof using a backup transaction monitor to prevent transaction blocking was implemented as early as 1980 in the SDD-1 distributed database system ; another implementation is reported. A replicated transaction coordinator to provide a non-blocking commit service has also been described. Our paper deals with the case of replicating transaction managers in the context of standards compliant Java application servers (J2EE servers).There are several studies that deal with replication of application servers as a mechanism to improve availability In , the authors precisely describe the concept of an exactly once transaction (e-transaction) and develop server replication mechanisms; their model assumes stateless application servers (no session state is maintained by servers) that can access multiple databases.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Their algorithm handles the transaction commitment blocking problem by making the backup server take on the role of transaction coordinator. As their model limits the application servers to be stateless, the solution cannot be directly implemented on stateful server architectures such as J2EE. The approach described in specifically addressed the replication of J2EE application servers, where components may possess session state in addition to persistent state stored on a single database implements the framework described in, therefore we concentrate our discussion on the implementation details of only). The approach assumes that an active transaction is always aborted by the database whenever an application server crashes. Our approach assumes the more general case of access to multiple databases; hence two phase commitment (2PC) is necessary. Application server failures that occur during the 2PC process do not always cause abortion of active transactions, since the backup transaction manager can complete the commit process.

Exactly once transaction execution can also be implemented by making the client transactional, and on web-based e-services, this can be done by making the browser a resource which can be controlled by the resource manager from the server side, as shown . One can also employ transactional queues to gain a similar result. In this way, user requests are kept in a queue that are protected by transactions, and clients submit requests and retrieve results from the queue as separate transactions. As a result, three transactions are required for processing each client request and developers must construct their application so that no state is kept in the application servers between successive requests from clients. The approach presented in guarantees exactly once execution on internet-based e-services by employing message logging. The authors describe which messages require logging, and how to do recovery on the application servers. The approach addresses stateful application servers with single database processing without replicating the application servers.

IMPLEMENTATION:

Replication with a single server:

By replicating state (resource managers) an application server may continue to make forward progress as long as a resource manager replica is correctly functioning and reachable by the application server. RDBMSs A and B are now replicated (replicas A1, A2 and B1, B2). Proxy resource adaptors (JDBC driver and XAResource interface) have been introduced (identified by the letter P appended to their labels in the diagram; note that for clarity, not all arrowed lines indicating communication between proxy adaptors and their adaptors have been-shown). The proxy resource adaptors reissue the operations arriving from the transaction manager and the containerto all replica resource managers via their resource adaptors.

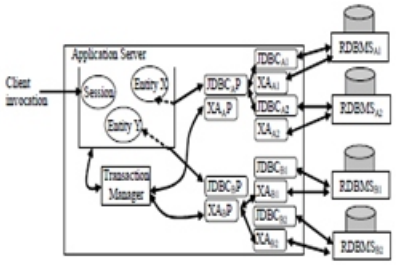

Fig.: An application server with state replication.

To ensure resource manager replicas remain mutually consistent, the resource adaptor proxy maintains the receive ordering of operation invocations when redirecting them to the appropriate resource adaptor replicas. This guarantees that each resource adaptor replica receives operations in the same order, thus guaranteeing consistent locking of resources across resource manager replicas.

Session State Replication:

The retry interceptor first identifies if this is a duplicated invocation by comparing the timestamp on the incoming client invocation with that in the timestamp log. If the invocation timestamp is the same as the timestamp in the timestamp log then the parameters held in the bean log are sent back to the client. If the invocation timestamp is higher than the timestamp in the timestamp log then the invocation is passed along the interceptor chain towards the bean. Upon delivery confirmation received from the replication service, the primary and backups update their bean and timestamp logs appropriately. Once such an update has occurred, the invocation reply is returned to the client.

Transaction failover management:

We assume container managed transaction demarcation. Via this approach to managing transactions the applicationdeveloper specifies the transaction demarcation for each method via the transaction attribute in a bean

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

deployment descriptor. Using this attribute a container decides how a transaction is to be handled. For example, if a new transaction has to be created for an invocation, or to process the invocation as part of an existing transaction (i.e., the transaction was started earlier in the execution chain). Based on this mechanism, a single invocation of a method can be: a single transaction unit (a transaction starts at the beginning of the invocation and ends at the end of the invocation), a part of a transaction unit originated from other invocation, or non transactional (e.g. the container can suspend a transaction prior to executing a method, and resume the transaction afterwards).

Load balancing:

The scheme described so far assumes a single primary that services all clients. To allow the scalability from clustering while benefiting from mid-tier replication, our scheme must be extendable to support load balancing for processing client requests. Extending our scheme to allow load balancing of client requests across a cluster of servers is straightforward. This is due to the nature of a session within J2EE: a session describes a relationship between a single client and a server, commonly denoted by the creation, usage and then deletion of a stateful session bean (instances of session beans are not shared by clients). To support load balancing, a client is attached to a session bean on a server. The choice of server is made in the normal way by the load balancer. This server is the primary, with all other servers acting as backups.

Jboss Implementation:

We use interceptors, management beans (MBeans), and Java Management Extensions (JMX) technologies to integrate our replication service into JBoss. This is the standard approach used when adding services to JBoss: interceptors intercept JBoss invocations while MBeans and JMX combine to allow systems level deployment of services that act upon intercepted invocations. This approach ensures that we do not disturb the functionality of existing services. The interceptors and associated services that implement our replication scheme.

Transaction failover management:

We assume container managed transaction demarcation. Via this approach to managing transactions the applicationdeveloper specifies the transaction demarcation for each method via the transaction attribute in a bean deployment descriptor.

Using this attribute a container decides how a transaction is to be handled. For example, if a new transaction has to be created for an invocation, or to process the invocation as part of an existing transaction (i.e., the transaction was started earlier in the execution chain). Based on this mechanism, a single invocation of a method can be: a single transaction unit (a transaction starts at the beginning of the invocation and ends at the end of the invocation), a part of a transaction unit originated from other invocation, or non transactional (e.g. the container can suspend a transaction prior to executing a method, and resume the transaction afterwards).The txinterceptor together with the transaction manager accommodates transactions within the application server. The replication service supports inter-replica consistency and consensus services via the use of JGroups. The replication service, retry interceptor, txinspector interceptor and the replica interceptor, implements our replication scheme.

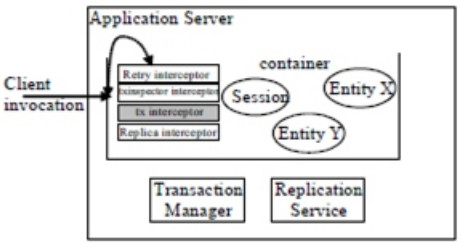

Fig:- Augmenting application server with replication service

Replication logic at the server side makes use of four in-memory logs that are maintained by the replicationservice: (i) current primary and backup configuration (group log), (ii) most recent state of session bean together with the last parameters sent back as a reply to a client invocation (bean log), (iii) invocation timestamp associated to most recent session bean state (timestamp log), (iv) state related to the progress of a transaction (transaction log). The replication service uses a single group via JGroups to ensure these logs are consistent across replicas.

PERFORMANCE EVALUATION:

Experiments were carried out to determine the performance of our system over a single LAN. Four experiments were carried out to determine the performance of the clustered (using JBoss clustering) and non-clustered approaches with and without state replication:

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

1) Single application server with no replication - To enable comparative analysis of the performance figures, an initial experiment was carried out to determine the time required to satisfy a client request issued to the application server using a single resource manager without state replication.

2) Single application server with state replication – Experiment 1 was repeated, with replica resource managers accessed by our resource adaptor proxy.

3) Clustered application server with no replication – Two application servers constituted the application server cluster with a single resource manager providing persistent storage.

4) Clustered application server with state replication – We repeated experiment 1 with replica resource managers ccessed by resource adaptor proxies from each of the application servers.

CONCLUSION:

In this paper, we have presented a new approach to improve the management features in a J2EE AS: adopting a component-based architecture for the middleware and for the management system. We selected Fractal model to represent the services in the middleware which allows having fine-grained management features. The same model is used to represent the management entities at different levels of abstractions thanks to the notion of domains. This allows aggregating and isolating different management policies. Thus, the management system and the managed services are represented in a uniform way to the manager which simplifies significantly the administrator tasks.

REFERENCES:

[1] S. Frolund and R. Guerraoui, “e-transactions: End-to-end reliability for three-tier architectures”, IEEE ransactions on Software Engineering 28(4): 378 - 395, April 2002
[2] www.jboss.org[3] A. I. Kistijantoro, G. Morgan, S. K. Shrivastava and M.C. Little, “Component Replication in Distributed Systems: a Case study using Enterprise Java Beans”, Proc. IEEE Symp. on Reliable Distrinbuted Systems (SRDS), pp. 89 - 98, Florence, October 2003,
[4] A. I. Kistijantoro, G. Morgan and S. K. Shrivastava, “Transaction Manager Failover: A Case Study Using JBOSS Application Server”, Proc. International Workshop on Reliability in Decentralized Distributed systems (RDDS), pp. 1555 - 1564, Montpellier, France, October 2006
[5] K. Birman, “The process group approach to reliable computing”, CACM , 36(12), pp. 37 - 53, December 1993.
[6] P. A. Bernstein, V. Hadzilacos and Nathan Goodman, “Concurrency Control and Recovery in Database Systems”, Addison- Wesley, 1987.
[7] M. C. Little, D. McCue and S. K. Shrivastava, “Maintaining information about persistent replicated objects in a distributed system”, Proc Int. Conf. on Distributed Computing Systems (ICDCS), pp. 491 - 498, Pittsburgh, May 1993
[8] M.C. Little and S K Shrivastava, “Implementing high availability CORBA applications with Java”, Proc. IEEE Workshop on Internet Applications (WIAPP ‘99), pp. 112 - 119, San Jose, July 1999
[9] P. Felber, R. Guerraoui, and A. Schiper, “The implementation of a CORBA object group service”, Theory and Practice of Object Systems, 4(2), pp. 93 - 105, April 1998
[10] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, “Consistent Object Replication in the Eternal System”, Theory and Practice of Object Systems, 4(2), pp. 81 - 92 April 1998
[11] R. Baldoni, C. Marchetti, “Three-tier replication for FT-CORBA infrastructures”, Software Practice & Experience, 33(18), pp. 767 - 797, May 2003
[12] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, “A Fault Tolerance Framework for CORBA”, Proc. of the IEEE Int. Symp. on Fault-Tolerant Computing (FTCS), pp. 150 - 157, Madison, USA, June 1999
[13] X. Zhang, M. Hiltunen, K. Marzullo, R. Schlichting, “Customizable Service State Durability for Service Oriented Architectures”, In Proc. of the Sixth European Dependable Computing Conf. (EDCC), pp. 119 - 128, Coimbra, Portugal, October 2006.
[14] Salas, J., Perez-Sorrosal, F., Patiño-Martínez, M., and Jiménez- Peris, “WS-Replication: a Framework for Highly Available Web Services”, In Proc. of the 15th Int. Conf. on World Wide Web (WWW), pp. 357 - 366, Edinburgh, Scotland, May 2006
[15] B. Roehm., “WebSphere Application Server V6 Scalability and Performance Handbook“, IBM Red Books, ibm.com/redbooks